Automatic Generation of Junit Test-Harnesses

Manfred Stock

ETH Zurich

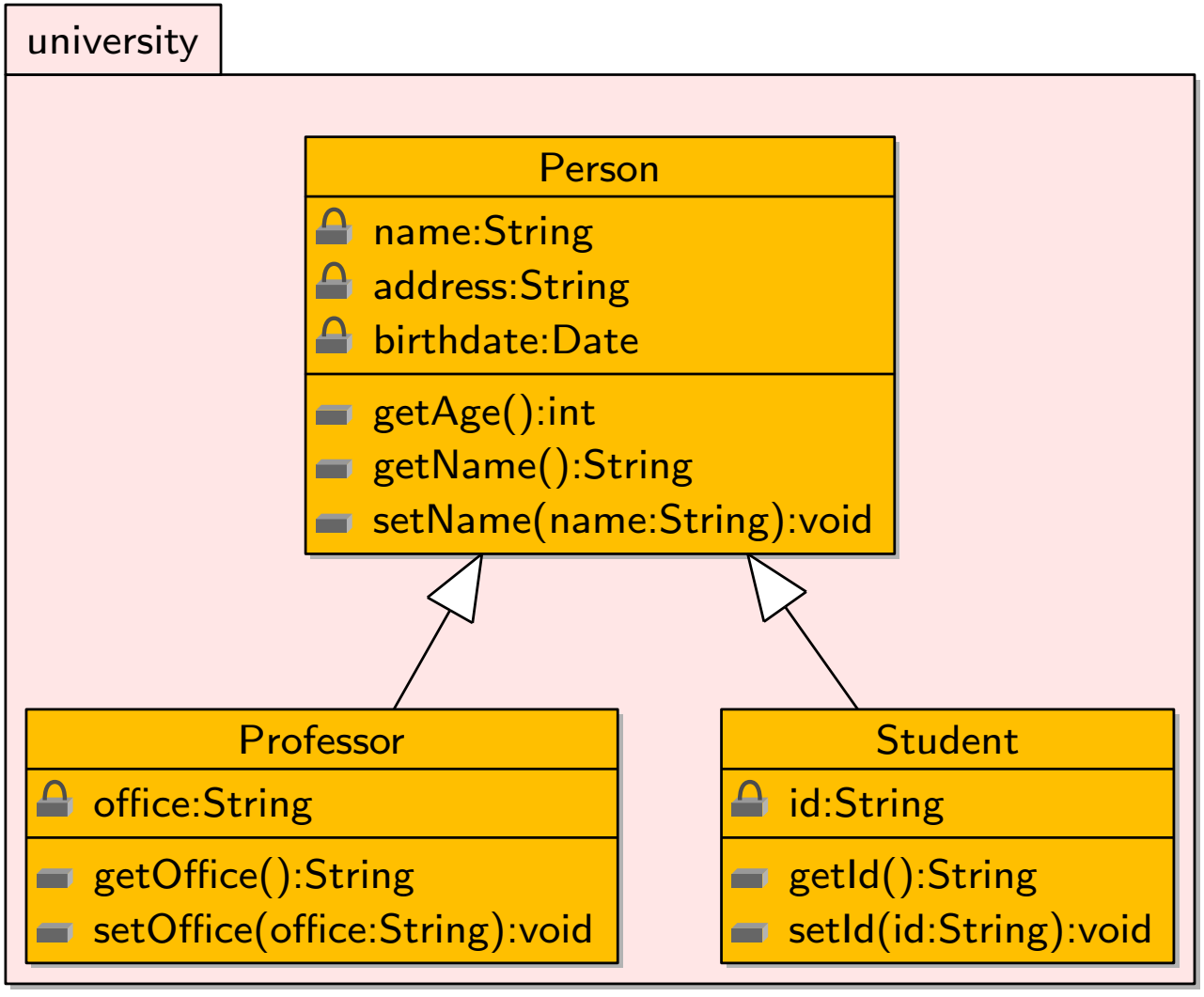
Semester Thesis Presentation
30.01.2007

Introduction

- Testing is an important activity during the development of software.
- The Information Security Group is developing a tool-chain supporting a security aware, UML-based software development process:
  - Access control specifications using SecureUML
  - Formal analysis using HOL-OCL
  - Automatic code-generation
- Generation of test-harnesses is an extension of this tool-chain

Outline

UML Class Diagrams



Object Constraint Language (OCL)

```
package university
context Person
  inv fields_nonnull: self.birthdate->notEmpty()
       and self.name->notEmpty() and self.address
      ->notEmpty()

context Person::getAge():int
  post positive_age: result >= 0

context Person::setName(name:String):void
  pre name_given: name->notEmpty()
  post name_set: self.name = name

context Student
  inv id_nonnull: self.id->notEmpty()

-- context Professor...
endpackage
```

Unit Testing

- Unit $\equiv$ A class in object oriented programming
- Tests the interfaces of a component/unit
- Testing compares intended and actual behaviours.
- Testing finds failures.
- Test-harness: A collection of software and test data configured for unit tests.
- According to E. W. Dijkstra, *"Testing can only reveal the presence, but never the absence of errors."*

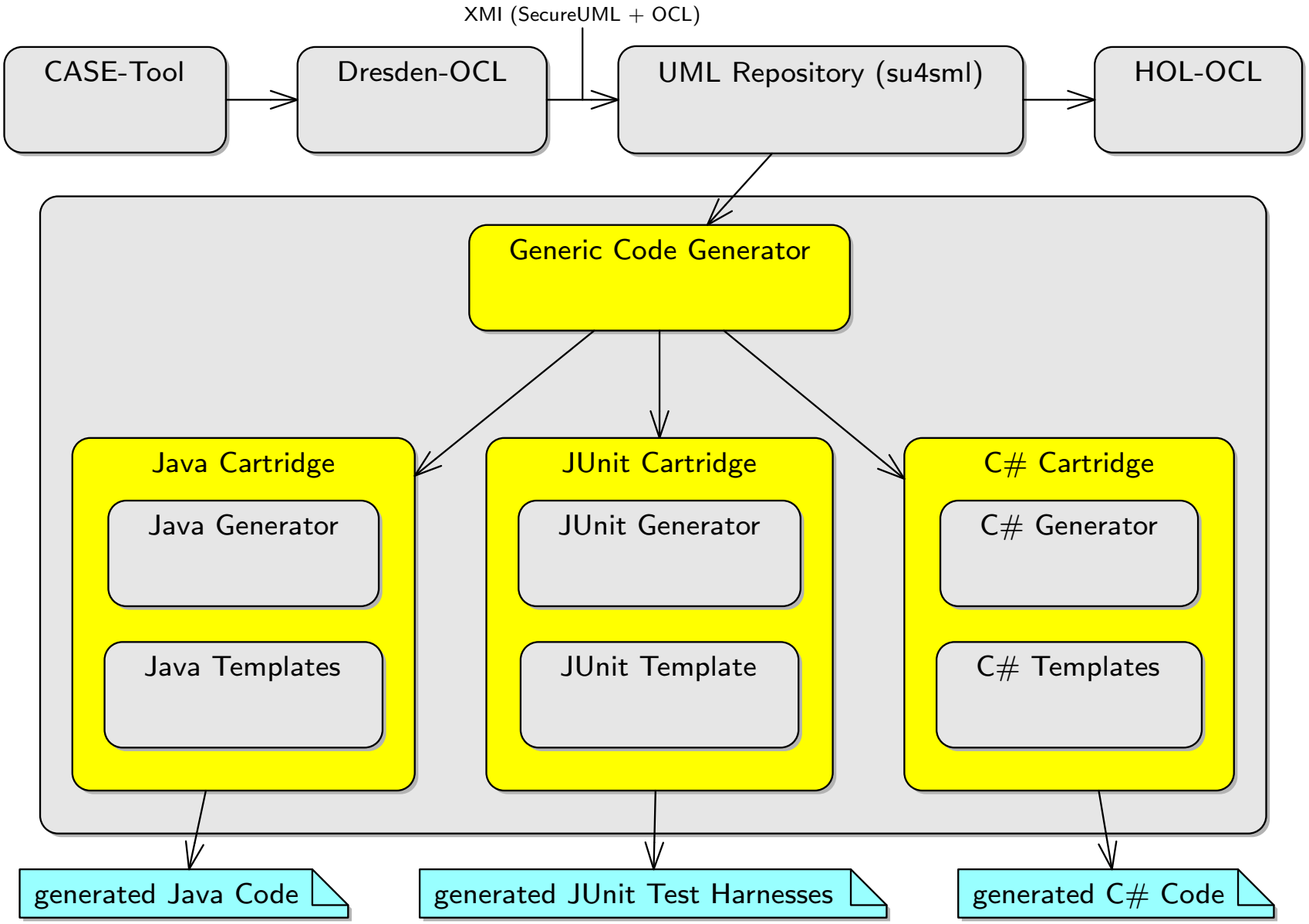
JUnit

- JUnit is the most famous unit testing framework for Java.
- JUnit provides the following:
  - Several annotations: `@Test`, `@Before`, `@BeforeClass`, `@After`, `@AfterClass`, `@Ignore` and others
  - Testrunners which run tests, collect results
  - Several assertions to compare actual and expected result
- To make use of JUnit, the following is required:
  - Actual tests:

```
import org.junit.Test;
public class JUnitSkeleton {
    @Test
    public void getAgeTest(){
        /* ... */
    }
}
```

  - Setup/teardown
  - Testdata
  - Test strategy

Environment

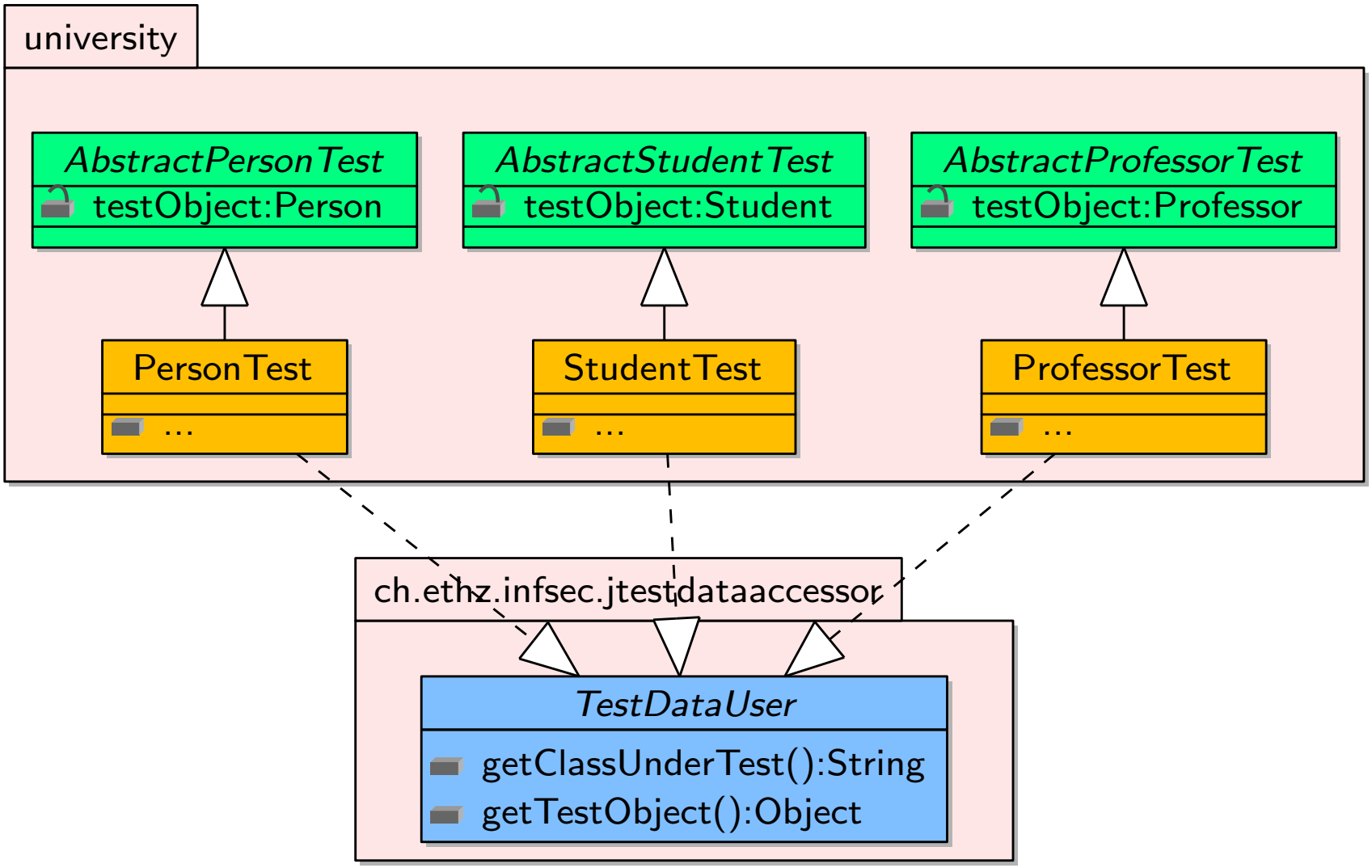


Testing Strategy

Generate 3 files for each class in the system:

- Abstract class:
  - Can/should be modified by the user
  - Setup/teardown methods, custom check methods
  - Custom `@Test` methods
  - Not overwritten during subsequent runs of code generator
- Concrete class:
  - Contains the actual `@Test` methods for each non-private method of the class and accesses test data
  - Overwritten during subsequent runs of code generator
- Stub for testdata:
  - Contains section for each non-private method of the class
  - Must be customized by the user
  - Not overwritten during subsequent runs of code generator

Testsuite Hierarchy



Directory Layout

The suggested directory layout was adopted from [Apache Maven]:

```
src/
|--main/
|   `--java/
|       `--university/
|           |--Person.java
|           `--...
`--test/
    |--java/
    |   `--university/
    |       |--AbstractPersonTest.java
    |       |--...
    |       |--PersonTest.java
    |       `--...
    `--resources/
        `--university/
            |--TestdataPerson
            `--...
```

Testdata Supply

- Testdata supplied using simple text file, similar to INI-files:

```
[getAge]
resulttype = int;          # Type of the result
setup = setup0();          # Setup the object
{
    result = 9;            # Expected result
    checker = EQUALS;      # Check of result
}

[setName]
inputtypes = String;       # Type of the input
setup = setup1();
{
    input = "Foo";         # Provided input
    checker = resultcheck0();
}
```

- There's also support for overloaded functions and exceptions.

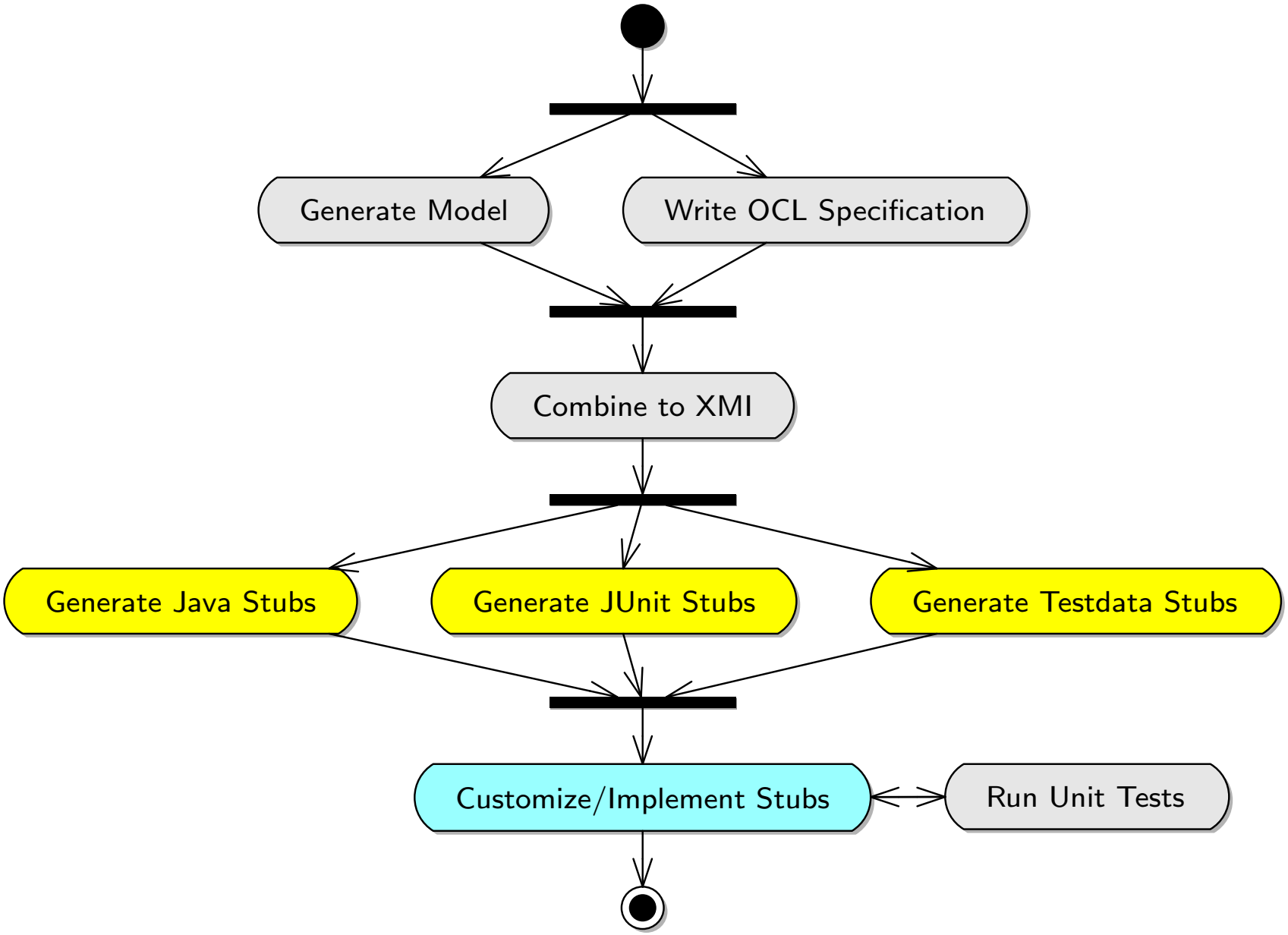
Usage of Testdata

- Implemented a library which allows access to testdata and provides some supporting functions for running unit tests
- When using the generated test harness, mostly invisible to the user

OCL Integration

- Generate code for the [Dresden OCL Standard Library]
- If there are OCL preconditions, they are used to check if the testdata fulfills them.
- If there are OCL postconditions or invariants, they are checked after executing the tested method.
- If precondition not satisfied, testdata does not get used, if postcondition/invariant not fulfilled, test failed

Workflow/Summary



Contributions of this Thesis

- Defined file format to provide testdata for unit tests
- Implemented Java library for access to this testdata
- Devised layout for test suite
- Wrote template and cartridge for code generator to generate test suite
- Several small improvements and extensions of code generator and existing Java template
- Conversion of internal OCL representation to Java-code which uses the Dresden OCL Standard Library (not yet completed)

Experiences

- Extension of existing tool-chain was fairly straightforward
- Automatic generation of test harnesses from model is feasible
- HOL-TestGen could be extended to generate testdata in a format which is usable by unit tests.

References


Apache Maven Project.
Introduction to the Standard Directory Layout.
http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html.


JUnit.
Testing Resources for Extreme Programming.
http://www.junit.org/.


Dresden OCL2 Toolkit.
OCL Standard Library.
http://dresden-ocl.sourceforge.net/introduction.html.